项目：Android系统中的系统异常及检测机制研究

**异常检测程序代码使用手册**

二○二三年八月

**目 录**

[1 性能异常检测代码使用说明 1](#_Toc135922415)

[1.1 异常检测流程 1](#_Toc135922416)

[1.2 代码使用方法 2](#_Toc135922417)

[1.2.1 模型格式转化 2](#_Toc135922418)

[1.2.2 性能异常检测APP主要代码结构 5](#_Toc135922419)

[1.2.3 运行界面展示 14](#_Toc135922420)

[2 功耗异常检测代码使用说明 17](#_Toc135922421)

[2.1 项目目录结构 17](#_Toc135922422)

[2.2 应用实现方法 17](#_Toc135922423)

[2.3 应用安装说明 22](#_Toc135922424)

[2.4 应用功能说明 23](#_Toc135922425)

[3 附录 24](#_Toc135922426)

本文将分别介绍性能异常和功耗异常应用的编译与部署方式和应用功能。

本文中代码部署的平台为一加 9 Pro 手机，其配置如表1所示。为了防止非原生系统可能对实验的干扰，本文禁用了设备上的 Bootloader 安全机制，并通过 fastboot 刷入了基于 Android 12 的 LineageOS 19.2 类原生操作系统，其 Linux 内核版本为 5.4.191，同时刷入了 Magisk 工具以获取设备上的 Root 权限。

表1 实验平台

|  |  |
| --- | --- |
| 相关平台 | 名称 |
| 手机型号 | OnePlus 9 Pro LE2120 |
| Android版本 | Android 12 |
| 操作系统版本 | Lineage OS 19-20221101-NIGHTLY-lemonadep |
| 内存大小 | 12GB(可用内存：10.99GB) |
| CPU版本 | 高通骁龙™ 888 移动平台 |
| CPU核心数目 | 8 |
| GPU版本 | Qualcomm, Adreno (TM) 660 |

# **性能异常检测代码使用说明**

本节会从性能异常检测程序代码检测流程与使用方法两个方面对相关代码使用进行说明。

* 1. **异常检测流程**

本项目的性能异常程序的检测流程如图1.1所示：

图示

描述已自动生成

图1.1异常检测流程

首先后台开启监控程序，监控程序周期性采集性能指标，当系统 Jank 比例超过阈值时，启用异常检测机制，首先将当前状态输入检测模块，根据 CPU、内存、GPU 三个决策树模型，判断有无因为后台抢占导致的前台应用性能异常，若存在CPU/内存后台抢占导致的异常，给出占用CPU与内存最多的应用程序列表，通知用户。当前台应用为已经建模的前台应用程序，使用基于特定前台的机器学习模型进行性能异常检测，否则使用整机机器学习模型进行性能异常检测。

* 1. **代码使用方法**
     1. **模型格式转化**

在使用APP项目前，首先需要考虑如何将之前训练的scikit-learn决策树模型部署到手机上，由于之前的决策树模型已经实现sklearn2pmml包导出为pmml模式，而pmml是数据挖掘的一种通用的规范，它使用统一的XML格式来描述我们生成的机器学习模型。当使用pmml时，需要两步的工作，第一步是将离线训练得到的模型转化为pmml模型文件,这已经在训练中完成，第二步是将PMML模型文件载入在线预测环境，进行预测。这两步都需要相关的库支持,pmml模型通过相关库的支持可以直接在计算机Java平台上运行，但是Android不提供自己的JAXB(Java Architecture for XML Binding (JAXB))运行时，通常不适合使用标准JAXB运行时间，如GlassFish Metro或EclipseLink MOXy。因此，org.dmg.pmml.pmml实例必须通过其他方式获得。 建议的解决方法是使用Java序列化来传输模型。这里通过pmml-android项目首先将pmml模型文件序列化为ser文件后在Android设备上执行，具体转化步骤如下：

1. 将生成的pmml文件中的xml头部的PMML-Version以及jpmml-model版本化为一致的4.3版本：

即将：

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<PMML xmlns="http://www.dmg.org/PMML-4\_4" xmlns:data="http://jpmml.org/jpmml-model/InlineTable" version="4.4">

修改为：

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<PMML xmlns="http://www.dmg.org/PMML-4\_3" xmlns:data="http://jpmml.org/jpmml-model/InlineTable" version="4.3">

1. sklearn模型转换pmml格式后，需要先进行序列化操作。此处需要下载pmml-android项目，项目地址为[https://github.com/loopGod/pmml-android，项目编译支持Android版本3.0](https://github.com/loopGod/pmml-android，项目目录如下，支持Android版本3.0)以上，jdk版本为版本8以下，版本9及以上需要自己集成JAXB，项目目录如下：

C:.

│ LICENSE.txt

│ pom.xml # 库引入文件

│ README.md

│

├─pmml-android # Android操作系统的PMML evaluator库

│ │ pom.xml # 库引入文件

│ │

│ ├─src

│ │ └─main

│ │ ├─android

│ │ │ AndroidManifest.xml #清单文件

│ │ │

│ │ └─java

│ │ └─org

│ │ └─jpmml

│ │ └─android

│ │ EvaluatorUtil.java # 序列化pmml文件

│ │

│ └─target

│ │ original-pmml-android-1.0-SNAPSHOT.jar

# 生成的JPMML-Android原始库

│ │ pmml-android-1.0-SNAPSHOT.jar

# 生成的JPMML-Android库

│ │

│ ├─classes

│ │ └─org

│ │ └─jpmml

│ │ └─android

│ │ EvaluatorUtil.class

│ │

│ ├─maven-archiver

│ │ pom.properties

│ │

│ └─maven-status目录

│ └─maven-compiler-plugin # maven编译相关

│ └─compile

│ └─default-compile

│ createdFiles.lst

│ inputFiles.lst

│

└─pmml-android-example # android=pmml示例程序

│ pom.xml

│

├─src

│ └─main

│ ├─android

│ │ │ AndroidManifest.xml #清单文件

│ │ │

│ │ └─resources # 资源文件

│ │ ├─…

│ ├─java

│ │ └─org

│ │ └─jpmml

│ │ └─android #主界面实现文件

│ │ MainActivity.java

│ │

│ └─pmml # 待使用pmml文件

│ model.pmml

│

└─target

│ AndroidManifest.xml

│ classes.dex

│ pmml-android-example-1.0-SNAPSHOT.apk #生成的示例apk

│ pmml-android-example-1.0-SNAPSHOT.ap\_

│ pmml-android-example-1.0-SNAPSHOT.jar

│ R.txt

│

├─classes

│ └─org

│ └─jpmml

│ └─android #相关类文件目录

│

├─generated-sources

│ ├─combined-assets

│ │ model.pmml.ser #生成的由pmml序列化之后的文件

│ │

│ └─r

│ └─org

│ └─jpmml

│ └─android

│ BuildConfig.java

│ R.java

│

├─…其他maven相关文件

进入根目录后，首先将自己的pmml文件放入pmml-android-example/src/main/pmml/目录下，在项目根目录下执行命令，使用mvn clean install编译项目，生成文件包括：pmml-android/target/pmml-android-1.0-SNAPSHOT.jar以及pmml-android-example/target/pmml-android-example-1.0-SNAPSHOT.apk，一个JAR库和一个示例APK。在目录pmml-android-example/target/generated-sources/combined-assets/下能够找到生成的序列化之后的.ser文件。

1. 将上述生成的pmml-android-1.0-SNAPSHOT.jar库放在之后的性能异常检测APP的app/libs目录下，使用implementate进行引入使用。将生成的多个ser序列化文件放在之后的性能异常检测APP的app/src/main/assets目录下，使得作为决策树模型作为资源文件部署到本地使用。
   * 1. **性能异常检测APP主要代码结构**

性能异常检测的APP项目的代码目录如下：

F:.

+---.gradle

+---.idea

+---app

| +---build #编译中间文件目录

| +---libs # 存放jar库文件

| |---AdbLib.jar #Adb的jar文件

| |---commons-codec-1.8.jar

| |---pmml-android-1.0-SNAPSHOT.jar #读取序列化之后的pmml文件

| \---src

| +---main

| | +---java

| | | \---com

| | | \---wwh

| | | \---updatemonitor

| | | |---ADBLIB.java #进行监控的主要实现

| | | |---AppInfo.java #APP信息类

| | | |---DecisionTreePredictor.java #决策树模型使

用类

| | | |---ForegroundService.java#前台Service父类

| | | |---MainActivity.java #应用主界面

| | | |---MonitorService.java#前台监控Service子类

| | | |---TextUpdater.java #前端显示信息更新类

| | \---assets # 存放序列化之后的资源文件

| | |---douyincpu.pmml.ser #抖音CPU异常检测文件

| | |---douyingpu.pmml.ser #抖音GPU异常检测文件

| | |---douyinmem.pmml.ser #抖音MEM异常检测文件

| | |---taobaocpu.pmml.ser #淘宝CPU异常检测文件

| | |---taobaogpu.pmml.ser #淘宝GPU异常检测文件

| | |---taobaomem.pmml.ser #淘宝MEM异常检测文件

| | |---yuanshencpu.pmml.ser #原神CPU异常检测文件

| | |---yuanshengpu.pmml.ser #原神GPU异常检测文件

| | |---yuanshenmem.pmml.ser #原神MEM异常检测文件

| | |---entirecpu.pmml.ser #整机CPU异常检测文件

| | |---entiregpu.pmml.ser #整机GPU异常检测文件

| | |---entiremem.pmml.ser #整机MEM异常检测文件

| | \---res # 存放资源文件

| | +---drawable

| | +---drawable-v24

| | +---layout #前端布局文件

| | +---mipmap-anydpi-v26

| | +---mipmap-hdpi

| | +---mipmap-mdpi

| | +---mipmap-xhdpi

| | +---mipmap-xxhdpi

| | +---mipmap-xxxhdpi

| | +---values

| | +---values-night

| | \---xml

| \---test #存放单元测试文件

| \---java

| \---com

| \---wwh

| \---updatemonitor

\---gradle

\---wrapper

本项目的开发Andoid版本为Android 12。项目可以导入到Android Studio中，编译为apk文件，或在项目主目录下执行gradle build命令编译，该命令会同时生成debug和release版本的应用，并保存到/app/build/outputs/apk目录下。编译完成后，可以将apk文件复制到设备上安装，或执行adb install命令安装。

APP主要完成的操作包括：各种性能异常数据的监控（通过ADBLIB执行shell命令作为一个前台Service完成监控）、性能异常检测（通过执行序列化之后的pmml决策树模型）完成。部分代码示例如下：

（1）监控代码示例：执行一次监控首先将命令输入终端，然后获取结果后续处理，设置监控时长每10s一次；

|  |
| --- |
| *//不在是一个循环，而是执行一个新的线程* @Override  **public void** doSomething() **throws** InterruptedException {  *// 循环执行状态读取  // 通过ADBLib执行指令* List<String> commands = **new** ArrayList<>();  *//获取时间* commands.add(**" echo \"$(date '+%s')time\""**);  *//获取前台应用package* commands.add(**" dumpsys window| grep mCurrentFocus"**);  *// 获取掉帧数量* commands.add(**" dumpsys SurfaceFlinger| grep missed"**);  *// 获取CPU和电池温度* commands.add(**" dumpsys thermalservice| grep Temperature"**);  commands.add(**" dumpsys thermalservice| grep 'Thermal Status'"**);  *// 获取Memory信息* commands.add(**" cat /proc/meminfo"**);  *// 获取GPU使用率与频率* commands.add(**" echo \"$(cat /sys/class/kgsl/kgsl-3d0/gpubusy)GpuUsage\""**);  commands.add(**" echo \"$(cat /sys/class/kgsl/kgsl-3d0/devfreq/cur\_freq)GpuFreq\""**);  *// 获取CPU频率* commands.add(**" echo \"$(cat /sys/devices/system/cpu/cpu0/cpufreq/scaling\_cur\_freq)Cpu0Freq\""**);  commands.add(**" echo \"$(cat /sys/devices/system/cpu/cpu1/cpufreq/scaling\_cur\_freq)Cpu1Freq\""**);  commands.add(**" echo \"$(cat /sys/devices/system/cpu/cpu2/cpufreq/scaling\_cur\_freq)Cpu2Freq\""**);  commands.add(**" echo \"$(cat /sys/devices/system/cpu/cpu3/cpufreq/scaling\_cur\_freq)Cpu3Freq\""**);  commands.add(**" echo \"$(cat /sys/devices/system/cpu/cpu4/cpufreq/scaling\_cur\_freq)Cpu4Freq\""**);  commands.add(**" echo \"$(cat /sys/devices/system/cpu/cpu5/cpufreq/scaling\_cur\_freq)Cpu5Freq\""**);  commands.add(**" echo \"$(cat /sys/devices/system/cpu/cpu6/cpufreq/scaling\_cur\_freq)Cpu6Freq\""**);  commands.add(**" echo \"$(cat /sys/devices/system/cpu/cpu7/cpufreq/scaling\_cur\_freq)Cpu7Freq\""**);  *// 获取网络相关情况* commands.add(**" dumpsys connectivity| grep 'Link speed'"**); ADBLIB adbl = **new** ADBLIB(**this**);  *// 获取结果* File filesDir = getApplicationContext().getFilesDir();  *//Init 执行一次* **thd** = **new** Thread(**new** Runnable() {  @Override  **public void** run() {  **if** (adbl.openStream(filesDir,**IP**)) {  Log.*d*(**"ADBInfo"**, **"Init"**);adbl.runCommand(commands,**appBeanList**);  HashMap info = **new** HashMap();  **try** {  Thread.*sleep*(2 \* 1000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  **if** (!adbl.**stream**.isClosed()) {  adbl.closeStream(info, **IP**, **false**,0);  }  Log.*d*(**"ADBLIB"**, **"Init Finished"**);  Log.*d*(**"ADBInfo"**, **"Init Finished"**);  }  }  });  **thd**.start();  *// 休眠10s* Thread.*sleep*(10 \* 1000);  **while** (**isRunning**) {  **thd** = **new** Thread(**new** Runnable() {  @Override  **public void** run() {  **if** (adbl.openStream(filesDir,**IP**)) {  Log.*d*(**"ADBLIB"**, **"StartMonitor: "** + filesDir);  adbl.runCommand(commands,**appBeanList**);  *// 获取内存信息* HashMap info = **new** HashMap();  getMemInfo(info);  Log.*d*(**"MAP"**, **"result: "** + info);  **try** {  Thread.*sleep*(5 \* 1000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  **if** (!adbl.**stream**.isClosed()) {  adbl.closeStream(info, **IP**, **true**,**foreW**);  }  Log.*d*(**"ADBLIB"**, **"results saved in"** + filesDir.toString());  Log.*d*(**"ADBInfo"**, **"results saved in"** + filesDir.toString()+**"\n\n"**);  }  }  });  **thd**.start();  *// 休眠10s* Thread.*sleep*(10 \* 1000);  }  } |

（2）匹配监控数据代码示例：通过正则表达式匹配输出，计算有效监控数据，下面展示了CPU利用率的计算：

|  |
| --- |
| *//匹配CPU信息计算CPU利用率 //region* String patternStr5 = **"cpu\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)"**; Pattern pattern5 = Pattern.*compile*(patternStr5); Matcher matcher5 = pattern5.matcher(results); **if** (matcher5.find()) {  **int** cpuUser = Integer.*parseInt*(matcher5.group(1));  **int** cpuNice = Integer.*parseInt*(matcher5.group(2));  **int** cpuSystem = Integer.*parseInt*(matcher5.group(3));  **int** cpuIdle = Integer.*parseInt*(matcher5.group(4));  **int** cpuIowait = Integer.*parseInt*(matcher5.group(5));  **int** cpuIrq = Integer.*parseInt*(matcher5.group(6));  **int** cpuSoftirq = Integer.*parseInt*(matcher5.group(7));  **int** cpuSteal = Integer.*parseInt*(matcher5.group(8));  **int** cpuGuest = Integer.*parseInt*(matcher5.group(9));  **int** cpuGnice = Integer.*parseInt*(matcher5.group(10));  **parsedResults**.put(**"cpuUser"**, matcher5.group(1));  **parsedResults**.put(**"cpuNice"**, matcher5.group(2));  **parsedResults**.put(**"cpuSystem"**, matcher5.group(3));  **parsedResults**.put(**"cpuIdle"**, matcher5.group(4));  **parsedResults**.put(**"cpuIowait"**, matcher5.group(5));  **parsedResults**.put(**"cpuIrq"**, matcher5.group(6));  **parsedResults**.put(**"cpuSoftirq"**, matcher5.group(7));  **parsedResults**.put(**"cpuSteal"**, matcher5.group(8));  **parsedResults**.put(**"cpuGuest"**, matcher5.group(9));  **parsedResults**.put(**"cpuGnice"**, matcher5.group(10));  Log.*d*(**"ADBInfo"**, **"cpuUser: "** + matcher5.group(1));  Log.*d*(**"ADBInfo"**, **"cpuNice: "** + matcher5.group(2));  Log.*d*(**"ADBInfo"**, **"cpuSystem: "** + matcher5.group(3));  Log.*d*(**"ADBInfo"**, **"cpuIdle: "** + matcher5.group(4));  Log.*d*(**"ADBInfo"**, **"cpuIowait: "** + matcher5.group(5));  Log.*d*(**"ADBInfo"**, **"cpuIrq: "** + matcher5.group(6));  Log.*d*(**"ADBInfo"**, **"cpuSoftirq: "** + matcher5.group(7));  Log.*d*(**"ADBInfo"**, **"cpuSteal: "** + matcher5.group(8));  Log.*d*(**"ADBInfo"**, **"cpuGuest: "** + matcher5.group(9));  Log.*d*(**"ADBInfo"**, **"cpuGnice: "** + matcher5.group(10));  **int** now = cpuUser + cpuNice + cpuSystem + cpuIdle + cpuIowait + cpuIrq + cpuSoftirq + cpuSteal + cpuGuest + cpuGnice;  *interval* = now - *cpuLast*;  **double** cpuUsage = 1 - (cpuIdle - *cpuLastIdle*) / (*interval* + 0.0);  *cpuLast* = now;  *cpuLastIdle* = cpuIdle;  *sinterval* = **true**;  **parsedResults**.put(**"cpuUsage"**, String.*valueOf*(cpuUsage));  Log.*d*(**"ADBInfo"**, **"cpuUsage: "** + String.*valueOf*(cpuUsage)); } *//计算前台CPU平均占用率* String patternStrFrontCpuUsage = *pid* + **" \\(([a-zA-Z]|\\.)\*\\) [a-zA-Z] \\d+ \\d+ \\d+ \\d+ -\\d+ \\d+ \\d+ \\d+ \\d+ \\d+ (\\d+) (\\d+) "**; Pattern patternFrontCpuUsage = Pattern.*compile*(patternStrFrontCpuUsage); Matcher matcherFrontCpuUsage = patternFrontCpuUsage.matcher(results); **if** (matcherFrontCpuUsage.find()) {  **parsedResults**.put(**"FrontApputime"**, matcherFrontCpuUsage.group(2));  Log.*d*(**"ADBInfo"**, **"FrontApputime: "** + matcherFrontCpuUsage.group(2));  **parsedResults**.put(**"FrontAppstime"**, matcherFrontCpuUsage.group(3));  Log.*d*(**"ADBInfo"**, **"FrontAppstime: "** + matcherFrontCpuUsage.group(3));  **int** FrontApptimeNow = Integer.*parseInt*(matcherFrontCpuUsage.group(2)) + Integer.*parseInt*(matcherFrontCpuUsage.group(3));  **if** (*sinterval*) {  **double** FrontAppCpuUsage = (FrontApptimeNow - *FrontApptime*) / (*interval* + 0.0);  **parsedResults**.put(**"FrontAppCpuUsage"**, String.*valueOf*(FrontAppCpuUsage));  Log.*d*(**"ADBInfo"**, **"FrontAppCpuUsage: "** + String.*valueOf*(FrontAppCpuUsage));  *sinterval* = **false**;  }  *FrontApptime* = FrontApptimeNow; } *//endregion* |

（3）使用决策树进行本地性能异常检测示例：通过使用序列化的ser模型文件，根据前台应用类型进行决策树模型推测。

|  |
| --- |
| **public class** DecisionTreePredictor {  **private** Context **mContext**;   **public** DecisionTreePredictor(Context context) {  **mContext** = context;  }   **public** Evaluator loadPmml(String pak,**int** a){  AssetManager assetManager = **mContext**.getAssets();  InputStream inputStream = **null**;  *//if(pak.equals("com.taobao.taobao"))* **try** {  **if** (pak.equals(**"com.taobao.taobao"**)) {  **if** (a == 1)  inputStream = assetManager.open(**"taobaocpu.pmml.ser"**);  **if** (a == 2)  inputStream = assetManager.open(**"taobaogpu.pmml.ser"**);  **if** (a == 3)  inputStream = assetManager.open(**"taobaomem.pmml.ser"**);  }**else if**(pak.equals(**"com.ss.android.ugc.aweme"**)) {  **if** (a == 1)  inputStream = assetManager.open(**"douyincpu.pmml.ser"**);  **if** (a == 2)  inputStream = assetManager.open(**"douyingpu.pmml.ser"**);  **if** (a == 3)  inputStream = assetManager.open(**"douyinmem.pmml.ser"**);  }**else if**(pak.equals(**"com.miHoYo.ys.mi"**)) {  **if** (a == 1)  inputStream = assetManager.open(**"yuanshencpu.pmml.ser"**);  **if** (a == 2)  inputStream = assetManager.open(**"yuanshengpu.pmml.ser"**);  **if** (a == 3)  inputStream = assetManager.open(**"yuanshenmem.pmml.ser"**);  }**else**{  **if** (a == 1)  inputStream = assetManager.open(**"entirecpu.pmml.ser"**);  **if** (a == 2)  inputStream = assetManager.open(**"entiregpu.pmml.ser"**);  **if** (a == 3)  inputStream = assetManager.open(**"entiremem.pmml.ser"**);  }  Log.*d*(**"model"**,**"Success: return model"**);  **return** EvaluatorUtil.*createEvaluator*(inputStream);  }**catch** (Exception e)  {  Log.*d*(**"model"**,**"Error: return model null"**);  e.printStackTrace();  **return null**;  }  }   **public int** predict(Evaluator evaluator,Map parsedResults)  {  Map<String, Double> data = **new** HashMap<String, Double>();  data.put(**"FrontAppCpuUsage"**,Double.*valueOf*((String) parsedResults.get(**"FrontAppCpuUsage"**)));  data.put(**"usedNativeMemPercent"**,Double.*valueOf*((String) parsedResults.get(**"usedNativeMemPercent"**)));  data.put(**"FrontAPPSystemMemPercent"**,Double.*valueOf*((String) parsedResults.get(**"FrontAPPSystemMem"**))/Double.*valueOf*((String) parsedResults.get(**"MemTotal"**)));  data.put(**"FrontAPPJavaHeapPercent"**,Double.*valueOf*((String) parsedResults.get(**"FrontAPPJavaHeap"**))/Double.*valueOf*((String) parsedResults.get(**"MemTotal"**)));  data.put(**"FrontAPPNativeHeapPercent"**,Double.*valueOf*((String) parsedResults.get(**"FrontAPPNativeHeap"**))/Double.*valueOf*((String) parsedResults.get(**"MemTotal"**)));  data.put(**"FrontAPPCodeMemPercent"**,Double.*valueOf*((String) parsedResults.get(**"FrontAPPCodeMem"**))/Double.*valueOf*((String) parsedResults.get(**"MemTotal"**)));  data.put(**"FrontAPPStackMemPercent"**,Double.*valueOf*((String) parsedResults.get(**"FrontAPPStackMem"**))/Double.*valueOf*((String) parsedResults.get(**"MemTotal"**)));  data.put(**"FrontAPPGraphicsMemPercent"**,Double.*valueOf*((String) parsedResults.get(**"FrontAPPGraphicsMem"**))/Double.*valueOf*((String) parsedResults.get(**"MemTotal"**)));  data.put(**"FrontAPPPrivateOtherMemPercent"**,Double.*valueOf*((String) parsedResults.get(**"FrontAPPPrivateOtherMem"**))/Double.*valueOf*((String) parsedResults.get(**"MemTotal"**)));  data.put(**"tempCPU0"**,Double.*valueOf*((String) parsedResults.get(**"tempCPU0"**)));  data.put(**"tempCPU1"**,Double.*valueOf*((String) parsedResults.get(**"tempCPU1"**)));  data.put(**"tempCPU2"**,Double.*valueOf*((String) parsedResults.get(**"tempCPU2"**)));  data.put(**"tempCPU3"**,Double.*valueOf*((String) parsedResults.get(**"tempCPU3"**)));  data.put(**"tempCPU4"**,Double.*valueOf*((String) parsedResults.get(**"tempCPU4"**)));  data.put(**"tempCPU5"**,Double.*valueOf*((String) parsedResults.get(**"tempCPU5"**)));  data.put(**"tempCPU6"**,Double.*valueOf*((String) parsedResults.get(**"tempCPU6"**)));  data.put(**"tempCPU7"**,Double.*valueOf*((String) parsedResults.get(**"tempCPU7"**)));  data.put(**"batteryTemp"**,Double.*valueOf*((String) parsedResults.get(**"batteryTemp"**)));  data.put(**"thermalStatus"**,Double.*valueOf*((String) parsedResults.get(**"thermalStatus"**)));  data.put(**"gpuUsage"**,Double.*valueOf*((String) parsedResults.get(**"gpuUsage"**)));  data.put(**"GpuFreq"**,Double.*valueOf*((String) parsedResults.get(**"GpuFreq"**)));  data.put(**"cpu0Freq"**,Double.*valueOf*((String) parsedResults.get(**"cpu0Freq"**)));  data.put(**"cpu1Freq"**,Double.*valueOf*((String) parsedResults.get(**"cpu1Freq"**)));  data.put(**"cpu2Freq"**,Double.*valueOf*((String) parsedResults.get(**"cpu2Freq"**)));  data.put(**"cpu3Freq"**,Double.*valueOf*((String) parsedResults.get(**"cpu3Freq"**)));  data.put(**"cpu4Freq"**,Double.*valueOf*((String) parsedResults.get(**"cpu4Freq"**)));  data.put(**"cpu5Freq"**,Double.*valueOf*((String) parsedResults.get(**"cpu5Freq"**)));  data.put(**"cpu6Freq"**,Double.*valueOf*((String) parsedResults.get(**"cpu6Freq"**)));  data.put(**"cpu7Freq"**,Double.*valueOf*((String) parsedResults.get(**"cpu7Freq"**)));  List<InputField> inputFields = evaluator.getInputFields();  *//过模型的原始特征，从画像中获取数据，作为模型输入* Map<FieldName, FieldValue> arguments = **new** LinkedHashMap<FieldName, FieldValue>();  **for** (InputField inputField : inputFields) {  FieldName inputFieldName = inputField.getName();  Object rawValue = data.get(inputFieldName.getValue());  FieldValue inputFieldValue = inputField.prepare(rawValue);  arguments.put(inputFieldName, inputFieldValue);  }   Map<FieldName, ?> results = evaluator.evaluate(arguments);  List<TargetField> targetFields = evaluator.getTargetFields();   TargetField targetField = targetFields.get(0);  FieldName targetFieldName = targetField.getName();   Object targetFieldValue = results.get(targetFieldName);  Log.*d*(**"model"**,**"target: "** + targetFieldName.getValue().toString() + **" value: "** + targetFieldValue.toString());  **int** primitiveValue = -1;  **if** (targetFieldValue **instanceof** Computable) {  Computable computable = (Computable) targetFieldValue;  primitiveValue = (Integer) computable.getResult();  }  Log.*d*(**"model"**,**"result: "** + String.*valueOf*(primitiveValue));  **return** primitiveValue;  } } |

除了本地部署代码外，当模型部署在服务器端时，需要在服务器端建立一个Server,两者可以使用Socket连接，服务器端可以使用python实现加载pkl模型进行推测，也可以使用Java实现加载pmml模型进行推测。为了客户的隐私，一般选择将模型部署在本地，对于服务器部署这里不再赘述。

* + 1. **运行界面展示**

程序运行主界面如下图1.2 (a)(b)(c)(d)：

图形用户界面

低可信度描述已自动生成图形用户界面, 应用程序

描述已自动生成图形用户界面, 应用程序, Teams

描述已自动生成图形用户界面, 应用程序

描述已自动生成

(a) (b) (c) (d)

图1.2 程序运行主界面

打开主界面首先显示当前IP,使用USB连接手机与电脑，在cmd窗口执行adb root以及adb tcpip 5555命令，然后点击确定进入主界面，点击启用监控，会弹出提示框选择预测方式，使用本地性能检测或者服务器性能异常检测或者不使用检测，点击确定后即可进入性能检测界面，如图1.2(c)显示当前正在进行监控加本地性能异常监测。当检测系统掉帧率超过阈值时，进行性能异常检测，如果异常会给出提示，如图1.2(d)所示。

运行时相关信息也会在终端打印出来供调试使用，如图1.3(a)(b)相关的监控数据保存在csv表格文件/data/data/com.wwh.updatemonitor/files/result.csv以及txt文本文件/data/data/com.wwh.updatemonitor/files/result.txt中。

表格

中度可信度描述已自动生成

(a)

日历

描述已自动生成

(b)

图1.3 运行时控制台信息

# **功耗异常检测代码使用说明**

本功耗异常检测程序为一个Android应用，使用Java语言实现，应用功能有自动化数据采集，数据处理并输入模型，在发生异常后自动进行异常定位并通知用户。本章将介绍项目的目录结构、应用的代码实现和安卓方法，最后展示应用的功能和使用方法。

## **项目目录结构**

powermonitorapp/app

├─build # 编译中间文件目录

└── src/main

   ├── AndroidManifest.xml

   ├── assets # 资源目录

   ├── java/com

   │   ├── cgutman

   │   │   └── adblib # adblib库，用于adb连接

   │   └── example

   │   └── poweranomaly # 存放代码文件

   │   ├── AppInfo.java # APP信息类

   │   ├── BatteryServiceAdb.java # 监控类

   │   ├── ForegroundService.java # 前台Service父类

   │   ├── Inference.java # 深度学习模型类

   │   ├── MainActivity.java # 主界面

   │   ├── NotificationBroadcast.java # 通知类

   │   └── utils # 工具类

   │   ├── AdbLib.java # adb连接工具

   │   ├── AlarmItem.java # 每个 class 的 Alarm 数据类

   │   ├── Alarm.java # 每个应用的 Alarm 数据类

   │   ├── AlarmsDumpsys.java # Alarm 获取

   │   ├── AnomalyEventAdapter.java # 主界面列表显示

   │   ├── AnomalyEvent.java # 主界面列表

   │   ├── BatteryUtils.java # 处理batterystats的函数

   │   ├── Consumption.java # 每个应用的资源使用数据类

   │   ├── ExecLib.java # shell 代码执行工具

   │   ├── JobsDumpsys.java # JobScheduler 获取

   │   ├── Wakelock.java # Wakelock数据类

   │   └── WakelocksDumpsys.java # Wakelock 获取

   └── res # 存放资源文件

├─layout

│ └─activity\_main.xml # 界面文件

└─...

## **应用实现方法**

应用启动后，会定时采集系统数据，并输入到模型中。短时异常和长时异常采集的特征如下表2.1和表2.2所示，具体采集代码可参考BatteryServiceAdb.java中的doSomething函数，由于代码过长，此处不做展示。

表2.1 短时异常特征

|  |  |  |
| --- | --- | --- |
| 特征名 | 含义 | 单位 |
| brightness | 亮度 | 0~255 |
| CPUUtilityRate | CPU利用率 | 0~100% |
| CPUCluster0UtilityRate | CPU小核利用率 | 0~100% |
| CPUCluster1UtilityRate | CPU中核利用率 | 0~100% |
| CPUCluster2UtilityRate | CPU大核利用率 | 0~100% |
| services | 正在运行的服务个数 | / |
| foregroundApps | 正在运行的应用个数 | / |
| temperature | CPU温度 | 摄氏度 |
| foregroundConsumption | 前台应用平均耗电 | mA |
| backgroundConsumption | 后台应用平均耗电 | mA |
| CPUConsumption | CPU平均耗电 | mA |
| wifiConsumption | Wifi模块平均耗电 | mA |
| mobileRadioConsumption | 数据网络平均耗电 | mA |
| foregroundAppType | 前台应用类型 | 应用 Category 中的 10 种类型 |
| foregroundCPU | 前台应用 CPU 时间 | jiffies/s |
| backgroundCPU | 后台应用 CPU 时间 | jiffies/s |
| foregroundNetwork | 前台应用网络流量 | Kbytes/s |
| backgroundNetwork | 后台应用网络流量 | Kbytes/s |
| radioType | 网络类型 | 0(其它), 1(Wifi), 2(2G), 3(3G), 4(4G), 5(5G) |
| radioStrength | 网络信号强度 | dBm |

表2.2 长时异常特征

|  |  |  |
| --- | --- | --- |
| 特征名 | 含义 | 单位 |
| wakelock | 最长的 Wakelock 的持续时间 | ms |
| alarmTime | Alarm 触发次数 | / |
| alarmWake | Alarm 触发唤醒次数 | / |
| alarmTotal | 所有 Alarm 触发总时间 | ms |
| jobTime | Job 触发次数 | / |
| jobTotal | 所有 Job 触发总时间 | ms |
| AppCPU | 后台应用 CPU 时间 | jiffies/s |
| AppNetwork | 后台应用网络流量 | Kbytes/s |
| AppCPUConsumption | 后台应用平均 CPU 耗电 | mA |
| APPGPSConsumption | 后台应用平均 GPS 耗电 | mA |
| AppSensorConsumption | 后台应用平均传感器耗电 | mA |

采集数据后，会输入深度学习模型进行推理，深度学习模型类inference.java文件内容如下，在程序初始化时，会从assets资源目录获取模型文件并加载到内存中，输入数据后，会先对数据进行归一化处理，然后输入模型并返回根据模型输出返回是否发生异常。

1. **public** **class** Inference {
2. **private** **static** Module powerModel = **null**;
3. **private** **static** **final** String PTL = "FCN\_mobile\_0308\_99.ptl";
5. **public** Inference(Context context) { # 加载模型
6. **try** {
7. powerModel = LiteModuleLoader.load(assetFilePath(context));
8. } **catch** (IOException e) {
9. e.printStackTrace();
10. }
11. }
13. **private** **static** String assetFilePath(Context context) **throws** IOException { # 读取模型文件
14. File file = **new** File(context.getFilesDir(), PTL);
15. **if** (file.exists() && file.length() > 0) {
16. **return** file.getAbsolutePath();
17. }
19. **try** (InputStream is = context.getAssets().open(PTL)) {
20. **try** (OutputStream os = **new** FileOutputStream(file)) {
21. **byte**[] buffer = **new** **byte**[4 \* 1024];
22. **int** read;
23. **while** ((read = is.read(buffer)) != -1) {
24. os.write(buffer, 0, read);
25. }
26. os.flush();
27. }
28. **return** file.getAbsolutePath();
29. }
30. }
32. **private** **static** **float**[] standardization(**float**[] raw) { # 输入数据归一化
33. **double**[] mean = **new** **double**[]{80.70466155810983, 0.3834093231162197, 0.4846203703703704, 0.4877180715197956, 0.49488633461047254, 0.49657024265644956, 0.29680427841634743, 0.3034645593869732, 0.3104651979565773, 0.21537835249042148, 76.96455938697318, 1.526500638569604, 64.27806800766282, 231.12416985951467, 197.16242017879952, 248.51270753512134, 101.07416347381864, 107.14663473818646, 211.67614942528735, 1797.8280332056195};
34. **double**[] std = **new** **double**[]{40.55143451982902, 0.22762060873803783, 0.16332114625380245, 0.176409572254452, 0.17979844819025106, 0.18272702384915238, 0.2980531995757685, 0.29453497184198757, 0.2909018792009807, 0.3086784144613514, 9.741110269155532, 0.49929722225884987, 8.998853758859752, 99.47813339170364, 226.68672666393914, 239.41424574275484, 65.93317598281953, 205.62307526955973, 200.15359060045884, 14970.808682094874};
35. **float**[] data = **new** **float**[20];
36. **for** (**int** i = 0; i < 20; i++) data[i] = (**float**) ((raw[i] - mean[i]) / std[i]);
37. **return** data;
38. }
40. **public** **static** **int** run(**float**[] data) { # 模型推理
41. Tensor input = Tensor.fromBlob(standardization(data), **new** **long**[]{20}); # 数据归一化并转换为tensor格式
42. Tensor outputTensor = powerModel.forward(IValue.from(input)).toTensor(); # 输入模型
43. **float**[] scores = outputTensor.getDataAsFloatArray();
44. Log.d("inference", Arrays.toString(scores));
45. **if** (Float.isNaN(scores[0])) {
46. Log.d("inference", "bad data: " + Arrays.toString(data));
47. **return** 0;
48. }
49. **return** scores[0] > scores[1] && scores[0] > scores[2] ? 0 : (scores[0] < scores[1] && scores[1] > scores[2] ? 1 : 2); # 根据模型输出判断是否异常
50. }
51. }

若发生异常，检测应用会判断导致异常的应用，并通知用户。异常定位的代码实现可参考BatteryServiceAdb.java中doSomething函数的最后部分，以下为部分内容:

1. **if** (modelOut == 1) { // CPU 异常
2. Log.d(TAG, appConsumption.toString());
3. Log.d(TAG, appCPU.toString());
4. List<Map.Entry<String, Consumption>> consumptionList = **new** LinkedList<>(appConsumption.entrySet()); // 对使用 CPU 的应用按照CPU耗电量排序
5. consumptionList.sort(Map.Entry.<String, Consumption>comparingByValue(Comparator.comparingDouble(v -> v.cpu)).reversed());
6. **double** sumCPUConsumption = appConsumption.values().stream().mapToDouble(v -> v.cpu).sum();
7. **double** cur = 0.0;
8. List<String> anomalyApp = **new** ArrayList<>();
9. **for** (Map.Entry<String, Consumption> entry : consumptionList) {
10. **if** (cur < sumCPUConsumption \* 0.8) {  // 选择前k个使用了80%资源的应用
11. cur += entry.getValue().cpu;
12. anomalyApp.add(entry.getKey().replaceFirst("u0a", "10"));
13. } **else** **break**;
14. }
15. notifyUser(1, anomalyApp.stream().distinct().collect(Collectors.toList()));   // 通知用户
16. } **else** **if** (modelOut == 2) { // 网络异常
17. Log.d(TAG, appNetwork.toString());
18. List<Map.Entry<String, Long>> networkList = **new** LinkedList<>(appNetwork.entrySet());
19. networkList.sort(Map.Entry.<String, Long>comparingByValue(Comparator.comparingDouble(v -> v)).reversed());   // 对应用按照网络流浪排序排序
20. **double** sumNetwork = appNetwork.values().stream().mapToDouble(v -> v).sum();
21. **double** cur = 0.0;
22. List<String> anomalyApp = **new** ArrayList<>();
23. **for** (Map.Entry<String, Long> entry : networkList) {
24. **if** (cur < sumNetwork \* 0.8) {  // 选择前k个使用了80%资源的应用
25. cur += entry.getValue();
26. anomalyApp.add(entry.getKey());
27. } **else** **break**;
28. }
29. notifyUser(2, anomalyApp);  // 通知用户
30. }

通知用户的代码实现可参考BatteryServiceAdb.java中的notifyUser函数，内容如下：

1. **private** **void** notifyUser(**int** type, List<String> appList) {
2. Log.d(TAG, "type" + type + appList.toString());
3. **try** { // 输出到文件中，以持久化
4. OutputStreamWriter outputStreamWriter = **new** OutputStreamWriter(getApplicationContext()
5. .openFileOutput("notify.txt", Context.MODE\_APPEND));
6. outputStreamWriter.write("type:" + type + ",list:" + appList + "\n");
7. outputStreamWriter.close();
8. } **catch** (IOException e) {
9. Log.e("Battery", "File write failed: " + e);
10. }
12. PendingIntent morePendingIntent = PendingIntent.getBroadcast(
13. **this**,
14. 0,
15. **new** Intent(**this**, NotificationBroadcast.**class**),
16. PendingIntent.FLAG\_UPDATE\_CURRENT | PendingIntent.FLAG\_IMMUTABLE
17. );
18. NotificationCompat.Builder notificationBuilder = **new** NotificationCompat.Builder(**this**)
19. .setSmallIcon(R.drawable.ic\_baseline\_add\_24)
20. .setAutoCancel(**true**)
21. .setContentTitle("发生功耗异常")
22. .setContentText("AnomalyApp 异常占用" + typeToStr(type))
23. .setGroupSummary(**true**)
24. .setChannelId("123")
25. .addAction(android.R.drawable.ic\_menu\_directions, "杀死", morePendingIntent)
26. .addAction(android.R.drawable.ic\_menu\_directions, "忽略", morePendingIntent); // 发送通知
27. NotificationManagerCompat notificationManager = NotificationManagerCompat.from(**this**);
28. notificationManager.notify(123, notificationBuilder.build());
29. addData("com.example.poweranomalyapp", type, System.currentTimeMillis()); // 显示到应用界面上
30. }

## **应用安装说明**

在编译项目前，需要预处理程序生成的standardization.txt文件中的两组归一化数据mean和std复制到inference.java文件的归一化函数中对应的数组中。将训练好的.pth模型放到powermonitorapp/app/src/assets资源目录中，并将对应文件名写入inference.java文件的字符串变量PTL中。

项目的TargetSDK为33，minSDK为30，编译该项目需要预先下载Android 33的SDK。项目可以导入到Android Studio中，编译为apk文件，或在项目主目录下执行。gradle build命令编译，该命令会同时生成debug和release版本的应用，并保存到/app/build/outputs/apk目录下。

编译完成后，可以将apk文件复制到设备上安装，或执行adb install命令安装。

项目使用adblib方式部署应用，在打开应用前，需要将手机通过USB连接到电脑上，并在电脑命令行执行adb root以及adb tcpip 5555命令，以打开手机的adb远程调试功能。

## 应用功能说明

应用提供开始异常检测和结束检测的按钮，并列出了过往被检测出的异常应用的类型和检测时间，如图2.1所示。用户可以查看发生过异常的应用，并对应用选择不同行为，如忽略该应用的异常、杀死应用、或者卸载应用。应用会记录用户的行为，并且在下一次异常时自动采取相同的行为。右下角是应用的设置按钮，在设置中用户可以设置应用的参数，包括设置检测周期、检测类型等。应用提供了白名单功能，用户也可以在设置中指定忽略某些应用造成的异常。

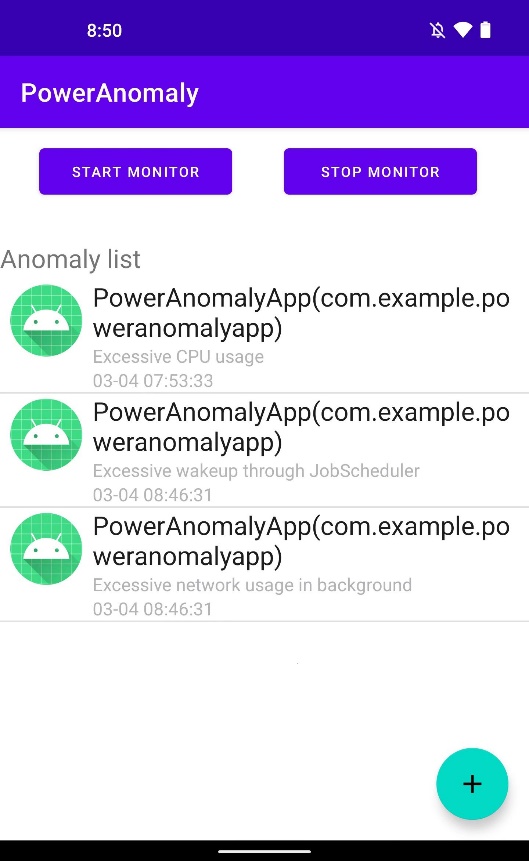


图2.1 应用首页

当异常发生后，应用会弹出通知，让用户选择是杀死应用或忽略，如图2.2。

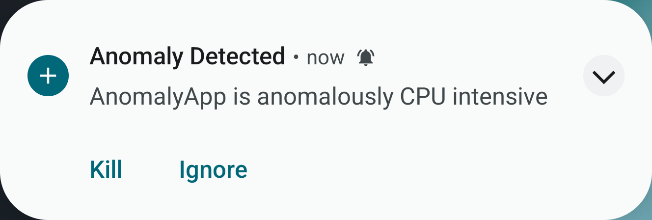


图2.2 异常通知

# **附录**

本阶段提交的文件清单如下所示：

.

├─code

│ ├─performance

│ │ ├─performance\_Decision # 性能异常检测机器学习模型

│ │ ├─pmml-android-master # 决策树模型序列化

│ │ ├─server # 远程服务器代码

│ │ └─updatemonitor # 性能异常检测APP代码

│ └─power

│ ├─power\_model # 功耗异常检测深度学习模型代码

│ ├─powermonitorapp # 功耗异常检测应用

│ └─powerdataacquisitionapp # 功耗异常数据采集应用

├─模型训练手册

├─代码使用手册 # 本文件

└─异常检测程序测试报告